UCRL-ID-141365

SafePatch

M. Kelley and S. Elko

October 1, 2000

U.S. Department of Energy

Lawrence
Livermore
National
Laboratory

DISCLAIMER

Work performed under the auspices of the U. S. Department of Energy by the University of California Lawrence Livermore National Laboratory under Contract W-7405-Eng-48.

SafePatch

Lawrence Livermore National Laboratory
PO Box 808 L-303
Livermore, CA 94551

October 2000
UCRL-ID-141365

Abstract

Authenticating and upgrading system software plays a critical role in information security, yet practical tools for assessing and installing software are lacking in today's marketplace. The SafePatch tool provides the mechanism of performing automated analysis, notification, distribution, and installation of security patches and related software to network-based computer systems in a vendor-independent fashion. SafePatch assists in the authentication of software by comparing the system's objects with the patch's objects. SafePatch will monitor vendor's sites to determine when new patches are released and will upgrade system software on target systems automatically. This paper describes the design of SafePatch, motivations behind the project and the advantages of SafePatch over existing tools.

Keywords: security, distributed, software management

Introduction

A serious threat to information resources is the inability to determine and maintain a known level of trust in operating system software. This threat can be minimized if systems are properly configured, use the latest software, and have the recommended security patches installed. However, the time and techniques required to assess and install recommended security patches on systems is considerable and too often neglected. This situation is further complicated by the fact that vendors have their own patch distribution and installation process. Though some vendors provide tools to assist with the installation process, these "solutions" (self-installing patches, installation utilities) fail in three critical ways: 1) the target system is not actually examined; 2) their solutions are vendor specific; and 3) they operate on a single host as opposed to a multi-host networked solution.

The SafePatch tool (formally referred as SSDS (Secure Software Distribution System)) provides automated analysis, notification, distribution, and installation of security patches and related software to network-based computer systems in a vendor-independent fashion. This allows network administrators to query, maintain, and upgrade the software integrity of hundreds of individual systems from a central point through an automated means. This centralized approach provides the following services for each targeted system:

- Rapid system software "trust" determination.
- Automated notification of new vendor security patches.
- Automated determination of patch applicability.

- Automated installation of security patches and critical system software.
- Ability to "back-out" installed patches, restoring a system's previous state.
- Collection of site-wide software statistics or metrics on patch status.

The process SafePatch uses to authenticate the software on a system is more reliable and secure than other vendor-specific tools. SafePatch compares the target system's objects with objects from the patches to determine what is actually installed and what needs to be installed. This approach ensures accurate reporting of a system's patch status. It also allows SafePatch to identify those objects that do not belong to either the original system distribution or to any released patches.

Motivation

System software plays a central role in information security. Most technological methods for securing system resources are critically dependent upon the system software. Defining access control lists (ACLs), properly setting up user and group accounts, and configuration of network services is useless if the software that is supposed to be enforcing these parameters are not performing what is expected. Short of controlling the physical access to a system, assessing and maintaining the integrity of system software in a networked environment is the first step in information security.

System software is constantly changing, making it difficult to maintain the integrity of a system. Often times, system software is security-flawed straight out of the box. Major network-wide assaults, such as the notorious 1988 Internet Worm attack, as well as a history of less publicized attacks, exploit these known security flaws to gain illicit access to systems. To their credit, vendors are often quick to issue security patches for vulnerable system files. However, even when vendors issue patches to fix a known vulnerability, a new release may inadvertently introduce further vulnerabilities into the system. This is common in large software companies because the teams that create new releases are often different than the teams that create the software patches. The multiplicity of security patches, couple with differing versions of the operating system, significantly complicates the software authentication effort.

Even if system software was certifiably "clean", software authentication efforts must also be concerned with the possibility of tampering during episodes of weak security management. A common method of compromising software security is to use a foothold on the system (*e.g.*, an unprotected user account) to modify key system files and compromise the system's defenses. Trojan horses are an example of this style of attack.

Vendors are aware of these issues and there is a push toward supplying the customers with "self-installing" patches or similar software installation to assist with the maintenance of software. However, these tools are highly vendor specific and vary wildly in their implementation and effectiveness. The tools we have encountered suffer from a common security flaw; they attempt to keep track of patches they have installed by building a "patch database" file. These tools can be easily fooled into reporting erroneous information because they make no attempt to survey the existing system files using secure cryptographic hashes or even ordinary checksums to ascertain what is actually installed. For example, assume a patch to fix a particular vulnerability has been installed using such a tool. Subsequently, an intruder replaces the fixed binary with a Trojan or older flawed version. Using a vendor tool to determine what is installed on the system may indicate the patch is already installed because the tool simply consults the "patch database". Solutions

that rely on a database are unreliable and unacceptable for determining the level of "trust" in operating system software.

Additionally, existing tools do not address the problem of maintenance in a heterogeneous network environment. In an environment where large mixtures of vendor systems are employed, the routine maintenance of software versions and patches is an administrative nightmare. Learning to operate and manage one vendor's set of software or patch management tools is grueling enough, let alone to do this for all the flavors of Unix and master their operational manuals. Including other popular operating systems further complicates the maintenance task even more. Is there any wonder why the installation of patches is to often neglected?

Software management tools are very much needed to support the assessment and authentication of system software on a network as well as installing and upgrading system software. Wouldn't it be nice to know exactly which of your 250 Unix systems are patched up-to-date, which are not, and what patches are needed for each system? Sadly, there are many organizations where an administrator of 250 systems could not determine this in a month's time, and certainly not in a manner that involved the actual examination of installed binary files. However, SafePatch enables an administrator to produce this information within hours of the request and will do so by actually examining the files present on these systems.

How much trust does a new network administrator place in the computer systems he/she just inherited? How much trust does an administrator place in a network recently experiencing suspicious activity? A responsible alternative to full network-wide system authentication would be to shut the machines down for a full re-install of their operating systems, along with the latest complement of security patches. This could represent weeks of service disruption, and the level of assurance it provides will tend to dwindle over time. More often than not, this simply doesn't get done. Again, this is why a software management tool supporting software authentication is needed.

A software management tool should also support software re-authentication on a regular basis, commensurate in frequency with the value of the resources being maintained on the systems. The SafePatch tool provides system administrators with a fast and highly automated method to authenticate system software, determine security patch versions and detect instances of subsequent tampering. In addition, SafePatch provides a convenient and secure means for automating the installation of required security patches and related system software. Information security demands this capability at its foundation.

SafePatch Architecture

A software management tool must be capable of 1) collecting patches, 2) determining which patches should be or have been applied to a system and 3) installing and possibly backing out patches. Patches can be collected from most vendors by downloading them directly from the vendor's ftp sites. To collect the latest releases, these ftp sites must be monitored on a regular basis.

Once the patches are downloaded to the local system, a software management tool must determine which patches should be or have been applied to a system. This is one of the most difficult tasks to automate in a software management tool. Each patch must be interpreted to determine the operating system type, version and architecture the patch applies to; how much memory and disk space is needed to install the patch; dependencies on other layered products or patches; and which files and directories are affected by the installation of a patch. To determine which patches are installed on a system, existing files

on a system must be compared with files contained in each patch. This process is commonly accomplished by manually reviewing a text file associated with each patch.

If the patch is applicable to the system, then the software management tool can install the patch, which usually entails following a set of instructions provided with the patch or executing a script. Sometimes the patch doesn't work as advertised or it interferes with other applications on the system, so the software management tool must also permit patches to be backed-out. Backing-out a patch is similar to the installation of a patch (*i.e.*, a set of instructions to follow or a script).

SafePatch largely automates the software management tasks described above. It accomplishes these tasks through two primary software components: the *SafePatch Command Center* and the *SafePatch Agent*. Figure 1 illustrates these components along with their interaction with the vendor's ftp sites and other network-based computer systems.

The SafePatch Command Center is the *central* service and resides on a single computer that interacts with several network-based computer systems. The network-based computer systems serviced by the SafePatch Command Center are referred to as target systems or Agents. The Command Center is responsible for monitoring a vendor's ftp site on a regular basis and collecting newly released patches. The SafePatch administrator can specify which vendor sites are to be monitored and which files to collect (*e.g.*, patches, readme, etc). In addition to acquiring and saving the patches themselves, each patch is processed to generate a vendor neutral machine-readable file. These vendor neutral files are referred to as "*patch specifications*" and contain information such as the operating system type, version, and architecture as well as the permissions and ownership for each file and directory manipulated by the patch. A cryptographic checksum for each file is also included in the patch specification to be used for file identification during the evaluation process described below. A patch specification file is built for each collected patch. By maintaining a complete history of all patches, SafePatch can determine what files are installed on a system and whether they are up-to-date. It is important for SafePatch to collect all patch revisions since vendors typically post only the latest revision of a patch. Eventually we hope that vendors will adopt a standard patch format or provide an adjunct for all of their patches (new as well as old patches).

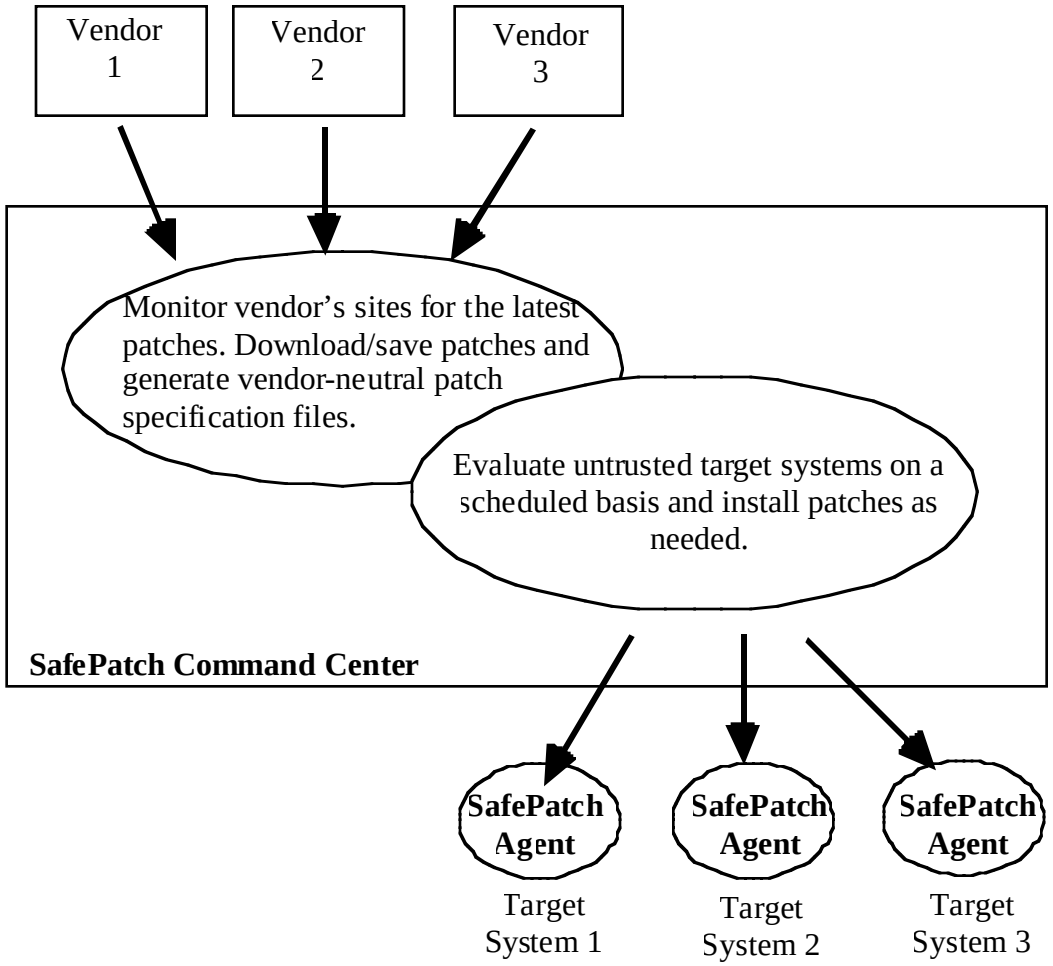

Figure 1

In addition to collecting patches from one or more vendors, the SafePatch Command Center is responsible for evaluating target systems. This includes downloading and installing those patches that will bring a system up-to-date. The SafePatch administrator has *full control* over the scheduling of target evaluations as well as the patch download and installation processes. They may request an evaluation immediately or schedule evaluations on a repeated basis. In short, a system administrator is in complete control and dictates all actions that SafePatch is to perform on a system.

The SafePatch Command Center controls the execution of an evaluation. To evaluate a system, the Command Center queries from the Agent its operating system, version, and architecture that it is running. It then collects all patch specifications corresponding to the information that was returned from the queries. From these patch specifications a list of directories and files manipulated by the patch is formed. The owner, group, permissions, and checksum (files only) for each file or directory on the list is checked against the owner, group, permissions, and checksums of the respective directory or file on the target system. This check permits SafePatch to determine which patches are actually installed on the target system without relying on the system's local database. From this information, SafePatch can determine which patches need to be downloaded and installed on the target system to bring it up-to-date. The system administrator can then choose to have SafePatch install patches immediately after the evaluation process or at some later time. The system administrator can also choose not to have SafePatch download and/or install the patches but

rather simply generate a report indicating which patches are needed to bring the system up-to-date.

The SafePatch Agent responds to commands and requests initiated by the SafePatch Command Center. The Agent is a lightweight process and uses very little resources on the target host; the majority of the work is performed by the centralized SafePatch Command Center.

Secure communications between the Command Center and the Agents can be used to protect data from being tampering with and to authenticate services requested. These secure communications employ digital signatures and encryption techniques based on public/private key technology.

SafePatch Today and Tomorrow

Started as a proof-of-concept effort in April of 1996, SafePatch has flourished into a fully functional and extremely powerful administrative tool that has been successfully deployed within the DOE and the U.S. Air Force. Lawrence Livermore National Laboratory has developed and maintained an ftp server (`safepatch.llnl.gov`) containing a complete set of archived patches for each of the supported operating systems. For authentication reasons, it is recommended that user's have SafePatch acquire patches directly from this ftp site rather than the vendor's site.

The current version of SafePatch provides support for Solaris 2.5.1+ and RedHat Linux 6.0+ systems. Though each of these operating systems can function as a SafePatch Agent, the Command Center is currently limited to the Solaris operating system.

Since the primary development effort has concluded, it is envisioned that the SafePatch technology will eventually be transferred to an outside agency for further enhancements and maintenance. Because SafePatch was developed using the object-oriented paradigm, it would be relatively easy and straightforward to provide additional support for other variants of the Unix operating system.

Conclusion

Automated information processing is destined to play an increasingly important role in our lives, and it will become critically important to assure trust in these information systems. SafePatch can fulfill a central role in this assurance with a uniform solution to the automated authentication and maintenance of system software. SafePatch will serve to protect against threats to information resources and provide a high level of trust to the systems' users.